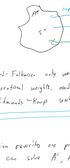
Lec23-flow-applications

Wednesday, November 6, 2024 4:24 AM

KT 7.5-7.7

Recall: Last time we showed that maximal s-t-max-flow problems were dual (flipped copy of) to s-t-min-cut problems.

Equivalence: Ford-Fulkerson treats augmenting paths until no s-t path exists in $G_{(f)}$, so then S is a cut (A^*, B^*).



$v(f^*) = \text{capacity}(A^*, B^*)$

(and f saturates all cut edges e: $f=c$ if e goes $A^* \to B^*$, $f=0$ if e goes $B^* \to A^*$)

Also, can't have higher $v(f')$

$\Rightarrow f^*$ is a max-flow & (A^*, B^*) is a min-cut

Note: Ford-Fulkerson only works with integer weights for $O(mC)$ time. For irrational weights, need to deal with cycle explicitly, but a variation Edmonds-Karp works in $O(nm^2)$ time.

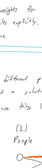
Reductions

A reduction examples one problem A as a different problem B', s.t. that if we can solve B', then we can get a solution of A.

Ex: We recruit problems as shortest path so we can use the Dijkstra on A^*

Maximum Bipartite Matching

- Set of people L & Tasks R
- Each person can only do certain jobs
- Bipartite graph



A matching assigns people to tasks

A maximum matching matches as many edges as possible

and maximum matching?

Formally: Given bipartite graph $G=(L \cup R, E)$, find $S \subseteq E \subseteq L \times R$ that is a matching and as large as possible

Note: Given L & R, is there need to find them though could using BFS

- S is a perfect matching if every vertex is matched
- Maximum $\leq$ perfect

$\uparrow$ as long as people $\leq$ tasks

can't match any bigger $\Rightarrow$ when nothing is shared

Reduce!: Connect problem into network flow problem to solve, when the solution to the flow problem is easily converted back into original matching problem.



turn the bipartite graph by points at across from L to R

Add vertex s pointing to all of L & pointing from all of R to t

Assign all edge capacity 1

Solve max-flow on new graph G' using Ford-Fulkerson

Analysis: Ford-Fulkerson augments with s-t paths each we being value 1.

Flow $\Rightarrow$ matching

Let M be the set of $L \to R$ edges on use in flow f.

M must be a matching because if two edges are used treating a node, the flow f would be violated since flow through a node is ≤ 1 by construction. Also $|M| = v(f)$ because each edge with a different unique node in L used, which must have 1 flow gets through it from s.

$\Rightarrow$ max-flow found by Ford-Fulkerson correspond to matching of same value

All matchings correspond to an integral flow

Given any matching simply push flow into all nodes of L with a scheduled partner, and push it out nodes of R with a scheduled partner.

$\Rightarrow$ A max-flow is also a maximum matching

Runtime:

Ford-Fulkerson takes $O(m'C)$ time, where m' is number of edges in the network $G' = (V', E')$ & $C = \sum_{e \in \text{out}(s)} c_e = |\text{out}(s)| = |L| = n$

$= O((n+L+n)n) = O(mn+n^2) = O(mn)$ time.

(edge in G goes to 2 in G')

the maximum bipartite matching.

Even though maximum bipartite matching doesn't look like a flow problem we can reduce it to a flow problem.

Max-flow extensions (LOOK)

The max-flow problem had just a single source s and target t.

What if multiple sources & targets

Circulation with demands has multiple sources S and multiple sinks T. (demand)

Supply = negative demand. $\forall v \in S$, let $d_v = -\text{(supply)}$

$\forall v \in T$, let $d_v = \text{demand}$



Feasible flow must have

$\sum_{v \in S} d_v + \sum_{v \in T} d_v = 0$.

Let $D = \sum_{v \in T} d_v$.

We want constraints:

$\forall e \in E,\ 0 \leq f(e) \leq c_e$

$\forall v \in V,\ f^{in}(v) - f^{out}(v) = d_v$

(0 for nodes not in S or T)

Reduction to max-flow

Add new super-source S^* & super-sink T^*

(S^*, s) has capacity $-d_s$

(t, T^*) has capacity d_t



Feasible circulation if and only if there is a flow of value D from S^* to T^*

Circulation with lower bounds

Suppose Pittsburg made a commitment to always ship at least 1 lbs of steel on the Pittsburgh-Philly line. Can we modify our algorithms?

New constraints:

$\forall e \in E,\quad \ell_e \leq f(e) \leq c_e$

$\forall v \in V,\quad f^{in}(v) - f^{out}(v) = d_v$.

We need to convert into ordinary circulation of demand problem.

Idea: Start by preallocating the flow demanded by ℓ_e as part of the demands by the ends.

Define the f_0 by setting $f_0(e) = \ell_e \quad \forall e \in E$

- meets capacity constraint but not demand constraint

Define $L_v = f_0^{in}(v) - f_0^{out}(v)$.

(normally we need $f^{in}(v) - f^{out}(v) = d_v$ for demand constraint)

New demand constraint: $f^{in}(v) - f^{out}(v) = d_v - L_v$

and new capacity constraint: $0 \leq f(e) \leq c_e - \ell_e$

(since we must send at the flow of ℓ_e already)

$\Rightarrow$ new standard circulation problem

Ex:



lower bound excess of shown solution



(Reduced) Transformation to equivalent problem w/o lower bounds

Spiral reduction:

Circulations with lower bounds

$\downarrow$

Circulations w/o lower bounds

$\downarrow$

max flow

Image segmentation



Given an image, what is foreground & what is background?

e.g. a tree against a blue sky.

Idea: Use color. If p_i is blue, likely to be background. If p_i is green, likely to be foreground.

More formally, have likelihoods a_i = likelihood pixel i is foreground, b_i = likelihood pixel i is background.

Coarse: If p_i is a foreground pixel, then nearby pixels are also likely foreground.

Idea: Encode image by undirected graph $G(V, E)$

pixel $\leftrightarrow$ edges between with neighbors

sky

except for things in sky

tree

penalty p_{ij} for putting i in foreground and j in background or vice versa

Image segmentation problem: Partition the set of pixels into two sets A & B to maximize $q(A,B) = \sum_{i \in A} a_i + \sum_{j \in B} b_j - \sum_{i \in A, j \in B} p_{ij}$

Minimum Cut problem: Partition the vertices of a directed graph into 2 sets A & B with $s \in A$, $t \in B$ to minimize weight of edges coming from A to B.

Difference:

- max vs. min
- source & sink
- objective function $q(A,B)$
- undirected vs. directed

Reduction: Make all edge into the directed edges with capacity p_{ij}:

Add source s with edge pointing to every pixel with capacity a_i

Add sink t with edge pointing from every pixel with capacity b_j

Want to maximize $q(A,B) = \sum_{i \in A} a_i + \sum_{j \in B} b_j - \sum_{i \in A, j \in B} p_{ij}$

Notice: Let $Q = \sum_{i \in V} (a_i + b_i) = \left(\sum_{i \in A} a_i + \sum_{j \in B} b_j\right) + \left(\sum_{i \in A, j \in B} p_{ij} + \sum_{i \in A, j \in B} p_{ij}\right)$

(every a_i & every b_j)

Same as minimizing $q'(A,B) = \sum_{i \in A} b_i + \sum_{j \in B} a_j + \sum_{i \in A, j \in B} p_{ij}$

Capacity of cut (A,B): $\underbrace{\sum_{i \in A} b_i}_{\text{cut edges } (i,t) \text{ with } i \in A}$ + $\underbrace{\sum_{j \in B} a_j}_{\text{cut edges } (s,j) \text{ with } j \in B}$ + $\underbrace{\sum_{i \in A, j \in B} p_{ij}}_{\text{cut edges in image}}$

(cut of pixels (A foreground, B background)

This tells the minimum cut in this network flow is exactly optimizing the foreground-background segmentation